UNIVERZA V LJUBLJANI

FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Peter Pavlinič

Program za odkrivanje ugodnih transakcij na borzi

DIPLOMSKO DELO

VISOKOŠOLSKI STROKOVNI ŠTUDIJSKI PROGRAM PRVE STOPNJE RAČUNALNIŠTVO IN INFORMATIKA

Ljubljana, 2014

UNIVERZA V LJUBLJANI

FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Peter Pavlinič

Program za odkrivanje ugodnih transakcij na borzi

DIPLOMSKO DELO

VISOKOŠOLSKI STROKOVNI ŠTUDIJSKI PROGRAM PRVE STOPNJE RAČUNALNIŠTVO IN INFORMATIKA

MENTOR: viš. pred. dr. Aleksander Sadikov

Ljubljana, 2014

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Program za odkrivanje ugodnih transakcij na borzi

Tematika naloge:

Diplomsko delo se ukvarja z učinkovitim iskanjem takih prodajno nakupnih poti med trgovalnimi elementi na trgu (borzah), ki so finančno ugodne za trgovca. V ta namen predstavi učinkovito podatkovno strukturo (graf) za hrambo podatkov o odnosih med trgovalnimi elementi in vrsto metod za iskanje po njej. Začne z neinformiranim izčrpnim preiskovanjem, se dotakne mogočega informiranega iskanja in ugotovi dovolj dober način za tolikšno zmanjšanje preiskovalnega prostora, da je mogoče izčrpno preverjanje vseh veljavnih poti. To je možno z uporabo algoritma za iskanje elementarnih ciklov v grafu.

IZJAVA O AVTORSTVU DIPLOMSKEGA DELA

Spodaj podpisani Peter Pavlinič, z vpisno številko **63110142**, sem avtor diplomskega dela z naslovom:

Program za odkrivanje ugodnih transakcij na borzi

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom viš. pred. dr. Aleksandra Sadikova;
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela;
- soglašam z javno objavo elektronske oblike diplomskega dela na svetovnem spletu preko univerzitetnega spletnega arhiva.

V Ljubljani, dne 15. avgusta 2014 Podpis avtorja:

Zahvaljujem se vsem, ki so me podpirali in mi stali ob strani.

Kazalo

Povzetek

Abstract

Poglavje 1 Uvod 1

Poglavje 2 Pristopi k reševanju problema 3

2.1 Iskanje s surovo silo 3

2.1.1 Iskanje v širino 3

2.1.2 Iskanje v globino 4

2.1.3 Naključno iskanje 4

2.2 Izdelava podatkovnega modela po meri 5

2.2.1 Tridimenzionalni nabor (matrika) 5

2.2.2 Slovar slovarjev s podatki o medsebojnih odnosih 5

2.3 Pristop z grafom 6

2.3.1 Neusmerjeni graf 7

2.3.2 Usmerjeni večkratno povezani graf 7

2.3.3 Usmerjeni graf 8

2.4 Informirano iskanje 8

2.5 Algoritem za iskanje elementarnih ciklov 9

2.5.1 Definicije 9

2.5.2 Delovanje Johnsonovega algoritma 10

2.5.3 Časovna zahtevnost Johnsonovega algoritma 10

Poglavje 3 Opis programa 13

3.1 Zgradba programa 13

3.2 Razredi programa 14

3.2.1 ApiParser 14

3.2.2 Commodity ... 14
3.2.3 Exchange ... 15
3.2.4 Main ... 15
3.2.5 PathFinder ... 15
3.2.6 Relations ... 16
3.2.7 Test ... 16
3.3 Decimal, float, zaokroževanje števil ... 16
3.4 Knizjnice ... 17
3.5 Aplikacijski vmesniki za borze ... 18
3.6 Uporabljeni algoritmi, razmislek ... 19
3.6.1 Idealne in slabše poti ... 19
3.6.2 Vsi cikli v grafu ... 19
3.6.3 Zaprti sprehod po grafu ... 20
3.6.4 Elementarni ali preprosti cikli ... 20
3.6.5 Filtriranje nepotrebnih ... 20
3.6.6 Ciklični zapis permutacij ... 21
3.6.7 Globina knjige naročil ... 21
3.6.8 Izris grafa ... 22
3.6.9 Serializacija - pickle ... 23
3.6.10 Največje težave pri implementaciji ... 23
Poglavje 4 Pohitritve, izboljšave, mogoče nadaljevanje ... 25
4.1 Boljši in nepopolni algoritmi za iskanje elementarnih ciklov ... 25
4.2 WebSocket ... 26
4.3 Knjižnica za graf, izvedena v jeziku C ... 26
4.4 Večnitnost ... 26
4.5 Pasivno/aktivno trgovanje ... 27
4.6 Agregatorji borznih indeksov ... 27
4.7 Trgovanje s posedovanjem več trgovalnih elementov ... 27
4.8 Uporabniški vmesnik in aplikacijski vmesnik ... 27

4.9 Programerska optimizacija 28

Poglavje 5 Sklepne ugotovitve 29

Definicije izrazov

izraz	angleško	razlaga
trgovalni element	tradable, commodity	Karkoli, s čimer je mogoče trgovati, bodisi valuta, vrednostni papirji, blago, delnice, obveznice, dragi kamni, surovine ali kripto valute.
elementarni cikel	elementary circuit/cycle, simple cycle	Cikel, v katerem se nobeno vozlišče razen zadnjega ne ponovi dvakrat.
Ločeni ali disjunktni elementarni cikel	distinct elementary circuit	Tak elementarni cikel, ki ni ciklična ali krožna permutacija drugega cikla v nekem prostoru.
podgraf	subgraph	Del grafa, ki je vsebovan v grafu.

Povzetek

Diplomsko delo se ukvarja z učinkovitim iskanjem prodajno-nakupnih poti med trgovalnimi elementi na trgu (borzah), ki so finančno ugodne za trgovca. V ta namen predstavlja učinkovito podatkovno strukturo (graf) za hrambo podatkov o odnosih med trgovalnimi elementi in vrsto metod za iskanje po njej. Delo se začne z neinformiranim izčrpnim preiskovanjem, se dotakne mogočega informiranega iskanja in ugotovi dovolj dober način za tako zmanjšanje iskalnega prostora, da je mogoče izčrpno preverjanje vseh veljavnih poti. To doseže z uporabo algoritma za iskanje elementarnih ciklov v grafu.

Ključne besede: borza, graf, preiskovanje, Johnsonov algoritem, arbitraža

Abstract

This thesis deals with efficient search of such chaining of trades on exchanges (stock markets), that are financially beneficial for the trader. I introduced efficient data structure (graph) for storage of data in form of relations between trading items and variety methods for search of beneficial trade paths on it. I started with uninformed exhaustive search, investigated possible informed searching methods and found a way to reduce search space so much that exhaustive evaluation of valid paths is possible. I achieved this with help of an algorithm for enumerating all elementary circuits in a graph.

Keywords: Exchange, Graph, Search, Johnson's algorithm, arbitrage

Poglavje 1 Uvod

Borzno trgovanje je ena prvih domen, v kateri se je uveljavilo elektronsko poslovanje z vsemi svojimi prednostmi, izmed katerih je daleč najpomembnejša hitrost poslovanja. Takšno poslovanje je odprlo veliko novih možnosti na tem področju. Danes lahko praktično vsak od doma in na poljubnem koncu sveta trguje z najrazličnejšimi valutami, delnicami, surovinami in drugimi pozicijami. Najbolj znano je visokofrekvenčno trgovanje, pri katerem se za dosego ugodnejših pogojev pri trgovanju izkorišča fizične zakasnitve omrežja. Strežniki za tako trgovanje so za dosego optimalnih pogojev po navadi fizično nameščeni v zgradbi borze. To veliko stane, vendar se vložki povrnejo z milijoni transakcij z razmeroma nizkim dobičkom. Zelo popularno je tudi trgovanje z uporabo napovednih modelov, kjer trgovci poskušajo z rudarjenjem iz podatkov in s strojnim učenjem predvideti gibanje tržne vrednosti blaga v prihodnosti. Vodilni na tem področju danes poleg zgodovinskih podatkov uporabljajo tudi zelo obsežno preiskovanje in avtomatično analizo poslovnih rezultatov, novic in celo socialnih omrežij. Pri opisanih dveh aktivnostih zaradi visokih finančnih in intelektualnih vložkov, potrebnih za uspeh, posamezniku ni lahko sodelovati. Obstaja pa še tretja možnost, tako imenovana arbitraža med trgovalnimi elementi. Gre za to, da trgovec najde zanj ugodno serijo transakcij. V ta namen izkorišča dva načina, ki prinašata ugodne nabore transakcij: nihanja tržnih cen in posledično zakasnitev reakcije trga ter morebitno nepovezanost, šibko povezanost ali likvidnost različnih delov trga. Zakasnitve tu niso nujno samo sekundne, ampak lahko trajajo tudi ure. V ekstremnih primerih je moč zaznati tudi tedne trajajoče zakasnitve. Vzroki za to so različni, od dolgotrajnega prenosa nakazil v tujino do nepopularnosti določenih dobrin na določenih koncih sveta.

Spreten trgovec danes lahko s pomočjo informacijske tehnologije uspešno odkriva take anomalije in jih izkorišča v svoj prid, obenem pa s takim početjem stabilizira trg. Na borzah je tako trgovanje zelo uveljavljeno in dobro sprejeto. Izredno neobičajno bi bilo, da bi tako anomalijo danes našel znotraj ene borze, saj jo trgovci takoj izkoristijo in s tem izničijo. Ta diplomska naloga predstavlja programsko opremo, izdelano za namen odkrivanja ugodnih trgovanj z arbitražo. Problema se loteva iz različnih zornih kotov in poskuša najti najboljši način za odkrivanje takih zaporedij trgovanj, da bodo za trgovca ugodne. Pri tem se poslužuje znanja, pridobljenega tekom študija in izkušenj pridobljenih pri arbitraži na stavnicah.

Cilj naloge je uspešno pridobiti večjo količino podatkov o trenutnem stanju trga ter jih analizirati z namenom odkrivanja ugodnih povezav. Končni cilj je simulacija trgovanja in ugotovitev, ali je na tak način mogoče pozitivno trgovati. Stranski produkt teh zagotovljeno »dobrih« transakcij bodo tudi podatki, uporabni za svetovanje, kateri posamezni nakupi na borzi so trenutno ugodni. To je mogoče doseči s primerjanjem različnih dobljenih trgovalnih poti na borzi. Taka pot običajno vsebuje pozitivne in negativne transakcije, vendar je končen seštevek pozitiven. To pomeni, da je iz več poti, ki se prekrivajo, mogoče ugotoviti, katera transakcija je ugodna. Dodaten cilj je izdelati program na modularen način, torej tako, da je dodajanje novih borz, novih trgovalnih elementov, novih provizij in tipov provizij ter iskalnih algoritmov enostavno.

Pri delu bo potrebno paziti na veliko pasti, ki jih tako trgovanje prinaša. Nekaj jih lahko predvidimo že vnaprej. Ažurnost in točnost podatkov bo bistvenega pomena. Napake v algoritmu lahko pomenijo veliko izgubo pri trgovanju. Upoštevati bo potrebno naš vpliv na borzo; če namreč izvedem nakup, nisem več le opazovalec, ampak sem aktivno spremenil stanje in cene na borzi. To za seboj potegne veliko premisleka o tem, kako in v kakšnih časovnih intervalih je dobro izvajati transakcije. Pri aktivnem trgovanju, kjer trgovec poda ponudbo oziroma naročilo, je potrebno upoštevati še dodatne efekte, kot sta količina in frekvenca trgovanja. Pri gradnji verig bo potrebno upoštevati tudi globino trga. Tržna cena nekega elementa na borzi pomeni, da en ali več trgovcev ponuja ali prodaja določeno količino tega elementa za določeno ceno. Ko pri nakupu ali prodaji presežemo to ceno, lahko naprej kupujemo po drugi, torej naslednji najboljši ceni. V verigi to pomeni, da je potrebno upoštevati najšibkejši člen. V tem primeru je to element z najnižjo globino pri določeni ceni. Ob vsem tem je potrebno voditi tudi evidenco vseh različnih provizij, ki jih je potrebno plačati bodisi ob prenosu sredstev iz denarnice na borzo ali med borzami. Provizije se za različne tipe trgovalnih elementov obračunavajo različno in imajo različne višine. Lahko so odvisne tudi od volumna preteklega trgovanja.

Za začetek bomo zaradi največje enostavnosti obdelovali podatke za valutno trgovanje in kripto valute. Iz davčnega in pravnega stališča je z njimi najmanj zapletov, za prikaz koncepta pa so več kot zadostne.

Poglavje 2 Pristopi k reševanju problema

Jedro problema pri tej nalogi je čim bolj natančno odkriti ugodne poti na trgu. Opisal ga bom tako, kot sem se ga lotil in kakor sem sproti razvijal ideje in tehnike za odkrivanje ugodnih poti. Drugih delov naloge, kot so pridobivanje podatkov, izris grafa in podobno se bom dotaknil kasneje, saj služijo le za podporo osnovnim funkcijam programa.

2.1 Iskanje s surovo silo

Najprej sem za občutek, da bi ugotovil, kakšne so razsežnosti problema in s koliko podatki delam, izdelal funkcijo, ki je delala permutacije nad trgovalnimi elementi. Preprosto je primerjala najprej prva dva, potem druga dva in tako naprej vse do najdaljših verig. Za gradnjo sem uporabljal kar binarni zapis števil. Število je dolgo toliko bitov, kolikor različnih trgovalnih elementov obstaja. Za vsako enico sem vzel pripadajoči trgovalni element in ga dodal v verigo. Potem sem preveril, če je veriga sploh veljavna. Veljavna je, če je zaporedne valute na borzah sploh mogoče zamenjati. Seveda je bila velika večina verig neveljavnih. Ta metoda je izredno naivna in potratna ter ima eksponentno časovno zahtevnost $O(n^n)$. Zato je neprimerna za karkoli drugega kot osnovno hitro testiranje na majhnem številu elementov.

Naslednji večji problem je bil, da še nisem imel podatkovne strukture, v kateri bi lahko shranjeval razmerja med pari valut. Zaradi tega sem delal veliko število poizvedb, kar je zaradi prevelikega števila zahtevkov pripeljalo do onemogočenja mojega naslova IP na eni od testnih borz.

2.1.1 Iskanje v širino

Ker je bilo splošno preiskovanje vseh permutacij popolnoma neučinkovito, sem poskusil na drug način. Poti sem začel iskati s preprostim algoritmom za iskanje v širino. Pri tem sem uporabljal že shranjene podatke, ki sem jih pred iskanjem pripravil v podatkovne strukture, ki jih bom opisal v nadaljevanju. To mi je omogočalo relativno hitro iskanje sosedov določenega trgovalnega elementa. S pojmom sosed je poimenovan element, v katerega je mogoče zamenjati tistega, iz katerega izhajamo. Glavna ideja tega načina je, da iz začetnega elementa ponovno iščemo točno ta element. Iskanju ne ustavimo, ko ga najdemo, saj želimo najti vse mogoče poti,

ne le ene, kot je opisano tudi v članku[3]. Te poti sproti shranjujemo, da jih kasneje lahko ocenimo. Ker želimo poznati vse poti, ki obstajajo med vsemi elementi, je ta postopek potrebno ponoviti za vsak element. Pri tem dobimo veliko poti, ki so dejansko že vsebovane v daljših poteh. Na nek način so krajše poti variacije daljših poti brez ponavljanja vozlišč.

Postopek je še vedno izredno počasen, saj zaradi razlogov, opisanih v poglavju 3.5.1, izdela ogromno poti, ki niso niti potrebne niti zaželene. Njegova časovna zahtevnost je zato še vedno prevelika za praktično uporabo. Problem povzroča tudi nepotrebno zasedanje pomnilnika, v katerem je v najslabšem primeru lahko tudi eksponentno število preiskovanih elementov (ker iskanja ne ustavimo, ko najdemo element). Poraba pomnilnika pa je odvisna tudi od vejitve oziroma povezanosti grafa. Najslabši teoretični primer se v realnosti ne pojavi, saj iz vsake valute ni mogoče menjati z vsako. V eksperimentalnih podatkih večina povezav med pari manjka. Poleg tega pa so preiskovani elementi majhni in današnje količine pomnilnika velike.

Iterativnega pristopa iskanja v širino nisem testiral, ker se mi za takšno uporabo ni zdel smiseln.

2.1.2 Iskanje v globino

Za vsak element lahko namesto iskanja v širino izvedemo algoritem, podoben iskanju v globino, ki se, enako kot prejšnji, ob prvem zadetku ne ustavi. A glede optimalnosti rešitve ni nič slabši od tistega v širino, ker v vsakem primeru išče vse rešitve in ne le prve, ki je lahko pri iskanju najkrajše poti v globino neoptimalna[8] (v smislu dolžine, ki pri našem problemu ne igra nobene vloge).

Iskanje v globino pa ima pred iskanjem v širino tudi prednost; porabi manj pomnilnika, saj mora naenkrat hraniti le toliko podatkov, kolikor jih vsebuje najdaljša mogoča pot po trgovalnih elementih.

2.1.3 Naključno iskanje

Naključno iskanje izvedemo tako, da po podatkih potujemo popolnoma naključno in prepotovane poti shranjujemo. Presenetljivo se obnese bolje kot prva opisana metoda, iskanje s surovo silo. V razumnem času namreč vrne razmeroma velik delež poti, ki je hitro uporaben za testiranje in iskanje pozitivnih. Če ga poganjamo dlje, se učinkovitost zelo hitro niža in asimptotično približuje celi rešitvi, ki jo lahko celo doseže, če je problem dovolj majhen.

2.2 Izdelava podatkovnega modela po meri

Za preiskovanja je bil potreben dober podatkovni model, ki bi vseboval vse preiskovane podatke. Kvalitete, ki jih pri tem opazujemo, so: hitrost vstavljanja elementov, hitrost iskanja elementov, hitrost iskanja elementu sosednjih elementov (trgovalnih parov), hitrost pridobivanja podatkov povezave med dvema elementoma in redundanca ter velikost podatkovne strukture.

Odločal sem se med tridimenzionalno matriko (izvedeno s seznami), kjer bi si dimenzije sledile po vrsti: prvi trgovalni element, drugi trgovalni element, borza, in pa med slovarjem slovarjev seznamov. V primeru slednjega bi bil prvi del izhodiščni element, drugi del pa trgovalni par tega elementa, vsi podatki o borzah in tečajih pa bi bili v zadnjem seznamu zapisani z več dimenzijami.

2.2.1 Tridimenzionalni nabor (matrika)

Večdimenzionalni seznam je pri vstavljanju in pri poizvedbi po indeksih zelo hiter in zato primeren za iteracijo, ki je naš osnovni proces. Poizvedba v smislu *relacija[eur][usd][borza_X]* nam vrne podatke o tem, koliko dolarjev dobimo za evre na borzi X, kakšne so provizije, kakšna je globina trga, časovni žig in druge potrebne podatke za obdelavo. Če bi podatkovno strukturo dejansko implementirali kot matriko, bi bila ta matrika zelo redka. V programskem jeziku Python tega problema ni, saj se seznami na račun hitrosti podaljšujejo avtomatično, torej z redko matriko ne zasedamo nepotrebnega prostora.

2.2.2 Slovar slovarjev s podatki o medsebojnih odnosih

Za izvedbo svoje podatkovne strukture sem uporabil slovar slovarjev, podatki pa se nahajajo v seznamu na koncu. Prednost tega je, da so povprečne poizvedbe izvedene v linearnem času, poleg tega pa nam ni potrebno uporabljati indeksov za hitrejše poizvedbe. Obstaja sicer pogoj: ključi v slovarju morajo imeti implementirano razpršitveno funkcijo »hashable«, vendar v mojem primeru to ni bil problem. V najslabšem primeru bi poizvedba porabila O(n) procesorskega časa. Enako velja za vstavljanje. Iteriranje je zaradi tega hitro in preprosto. Potrebno pa je omeniti, da kljub linearnemu času tu govorimo o nekajkrat počasnejših operacijah, kot so bile v seznamu, predvsem na račun razpršitvene funkcije.

Podatki o povezavi na koncu se nahajajo v seznamih predvsem zaradi hitrosti. Ker jih ni veliko in imamo nadzor nad njihovim ustvarjanjem, je preprosto dodeliti zaporedno mesto določenemu podatku in ga na tak način tudi brati. Iteracije, ki se bodo izvajale na tem seznamu,

so hitrejše kot v slovarju. Iterira pa se večkrat, predvsem pri ocenjevanju poti; najprej po borzah, potem pa še po vrednostih in globinah trga.

2.3 Pristop z grafom

Ko sem naredil lastno strukturo, se je izkazalo, da je to dejansko graf. Trgovalne enote so vozlišča, razmerja med njimi pa povezave. To me je navdušilo, saj je nad grafom mogoče izvajati vrsto preiskovanj z uveljavljenimi in hitrimi algoritmi. Na tem mestu sem moral sprejeti kompromis ali naj uporabim že obstoječo knjižnico in s tem pridobim številne nepotrebne metode in redundantne pomožne strukture, ki jih take knjižnice uporabljajo. Po pregledu obstoječih rešitev za analizo grafov v jeziku Python sem se prepričal, da bo knjižnica izpolnila potrebe in zahteve mojega programa. Vsaka od njih ima v primerjavi z drugimi določene prednosti in določene slabosti. Te bodo podrobneje opisane v poglavju 3.1.3. Izbral sem v jeziku Python izvedeno knjižnico networkx. Kljub temu da performančno zaostaja za drugimi, ki imajo algoritme izvedene v jeziku C, se je za moje potrebe odlično obnesla. Zasnovana je pregledno, smiselno in učinkovito ter ima dobra navodila za uporabo. Ob njihovem branju sem ugotovil, da je osnovna struktura grafa zasnovana zelo podobno kot moja struktura – s slovarjem slovarjev.

Ob branju dokumentacije sem posebno pozornost posvečal iskanju poti po grafu. Tu velja omeniti obilo algoritmov, ki najdejo najkrajšo. Večina jih temelji na ali uporablja algoritem Dijsktra, ki sem ga posredno uporabil tudi sam. Najprej sem se usmeril v preiskovanje, nato v barvanje grafa, potem pa v izpis vseh poti od točke do točke. Zelo obetavni so bili vpeti grafi, ki jih dobri algoritmi zgradijo v izjemno kratkem času. Če jim dodajamo nove robove, dobimo bazo ciklov, ki se mi je na prvi pogled zdela dobra za uporabo v mojem problemu. Cikel je namreč mogoče raztegniti v pot, konča pa se točno tam, kjer se je začel. To pomeni, da končaš v valuti, s katero si začel trgovati, s to razliko, da je imaš ob ugodnem poteku na koncu več kot na začetku.

Izkazalo se je, da baza ciklov ni dobra rešitev, saj ne zajema vseh poti, ki jih moramo obdelati. Baza ciklov je sestavljena iz osnovnih ciklov, iz katerih je mogoče z uporabo simetrične razlike sestaviti katerikoli Eulerjev graf (natančno enkrat uporabi vsako povezavo med vozlišči) [6]. Vendar tudi to niso vse poti, ki jih potrebujemo za rešitev problema.

Naslednja stvar, ki sem jo raziskoval, so bili algoritmi [1] za označevanje vseh ciklov v grafu. Želel sem jih pametno filtrirati in obdržati samo uporabne za moj namen. Zaradi izredno velike

časovne zahtevnosti tega postopka sem to idejo hitro opustil. Graf z ducatom trgovalnih elementov je zelo hitro dosegel milijonska števila ciklov. Natančnejši opis tega pristopa in časovna zahtevnost algoritma sta opisana v poglavju 3.1.5.2 in v poglavju 3.1.5.5.

Po tem sem raziskoval možnost uporabe algoritmov za iskanje zaprtih sprehodov po grafu (poglavje 3.1.5.3). Zaprti sprehod je dejansko cikel, ki pa lahko eno vozlišče obišče večkrat. Tu naletimo na težavo. Če lahko eno vozlišče obiščemo večkrat, potem lahko tak cikel razbijemo na dva krajša. Če je cikel mogoče razbiti na dva krajša, potem za nas ni uporaben, saj je vedno zaželeno imeti najkrajše možne cikle, ki lahko opišejo vse mogoče poti. Zakaj je tako, je razloženo v poglavju 3.1.5.1.

Ko sem raziskoval, kako bi zaprte sprehode pretvoril v meni uporabne cikle, sem našel algoritme za iskanje vseh preprostih ciklov v grafu. Preprost cikel je tisti, ki določeno vozlišče in določeno povezavo obišče le enkrat, razen seveda prvega vozlišča, kamor se na koncu vrne. Ločeni osnovni ali elementarni cikli so tisti, ki niso ciklične permutacije eden drugega. To pa je natančno to, kar iščemo. Za določevanje takih ciklov obstaja več algoritmov, ki so v večji meri evolucije prvega. Natančneje so opisani v poglavju 3.1.5.4 in 4.1. Dobljene cikle sem nato filtriral, da sem izločil krajše od treh vozlišč. Uporabljal sem namreč usmerjen ali digraf. Cikel razdalje dva pomeni, da upoštevamo menjavo na primer iz evra v dolar in nazaj v evro. Taka menjava bo vedno slaba, saj svoje poberejo provizije in razlike v nakupni ter prodajni ceni. Omeniti moram tudi zanke. Gre za tip povezave, ki jo lahko vsebujejo vsi spodaj opisani tipi grafov. V nekaterih primerih algoritmi to pravilno upoštevajo, v drugih jih je potrebno odstraniti. Mi se s tem ne bomo ukvarjali, ker zanke v tem kontekstu niso smiselne.

2.3.1 Neusmerjeni graf

Prva implementacija je bila izvedena z uporabo klasičnega neusmerjenega grafa. Njegova prednost je, da je enostavnejši in je na njem mogoče izvajati širok nabor operacij. Težava pa je v tem, da je potrebno vse podatke za trgovanje v obe strani hraniti na eni povezavi. Kadar je trgovanje mogoče samo v eno smer, moramo to posebej obravnavati kot izjeme pri shranjevanju in iskanju ciklov. Glavni algoritem, ki sem ga uporabljal, bi moral za iskanje osnovnih ciklov na takem grafu le-tega popraviti oziroma ga pred obravnavo preoblikovati v primerno obliko. Zaradi tega sem strukturo raje zamenjal z usmerjenim grafom. Potrebna je bila le predelava funkcij za pretvorbo in vstavljanje podatkov.

2.3.2 Usmerjeni večkratno povezani graf

Večkrat povezani, večpovezavni ali s tujko multi graf se najbolj izkaže pri predstavitvi podatkov. Dodatne povezave namreč lahko predstavljajo poljubno novo dimenzijo, v našem

primeru je to trgovalna borza. Evre namreč lahko v švicarske franke pretvarjamo na veliko borzah. Tak graf je lahko usmerjen ali neusmerjen. Bistvena slabost je, da večine operacij, ki podpirajo klasičen graf, na tem ni na voljo. Le nekaj je takih, ki niso na voljo zaradi tehničnih razlogov. Največkrat gre za to, da jih preprosto ni še nihče implementiral. V nekaterih primerih so spremembe minimalne, v drugih pa je potrebno dobro izkoristiti lastnosti grafov, da jih bodisi predelamo v primerno obliko bodisi algoritem spremenimo, da bo deloval na njih. Velikokrat je to izvedeno tako, da jih algoritem interno predela v drugačno obliko.

Druga slabost tega pristopa je, da je časovno veliko zahtevnejši od tistih, pri katerih borze ne označujemo z novimi povezavami, ampak vse obstoječe predstavimo šele znotraj povezave. Kompleksnost iskanja elementarnih ciklov se namreč z dodajanjem povezav povečuje veliko hitreje kot kompleksnost, če se problema lotimo tako, da vedno iteriramo po vseh borzah v povezavi. Slednji način nam poveča O kompleksnost zgolj za n-kratnik povprečnega števila borz v povezavi, pa še to le, če trgovanje omejimo na določene borze, ko je graf že sestavljen. Temu se izognemo tako, da borz, na katerih ne želimo trgovati, ne shranimo v obravnavani graf. Druge operacije zaradi takega shranjevanja niso počasnejše. Kompleksnost za iskanje elementarnih ciklov pri dodajanju povezave je težko merljiva, saj je močno odvisna od topologije grafa.

2.3.3 Usmerjeni graf

V končni rešitvi problema je obveljal prav usmerjeni graf, saj prinaša več prednosti glede na prej omenjena. Je kompromis, v katerem so podatki za iskanje, torej pari sosednjih valut v grafu, predstavljeni z največ dvema povezavama v paru. Omenjeno velja za primer, ko je trgovanje mogoče v obe smeri. To pripomore k manjšemu številu povezav in tako k hitrejšemu delovanju algoritma za iskanje neodvisnih elementarnih ciklov. Glavni problem, ki ima največjo časovno zahtevnost v programu, smo tako zmanjšali na najmanjše možno število podatkov. Vse ostale podatke shranjujemo kot atribute na povezavi, kjer so kompleksnosti, ki jih srečujemo, veliko manjše.

2.4 Informirano iskanje

Pri iskanju ugodnih poti je pred začetkom izvajanja zelo težko ugotoviti, katere poti bi bile lahko boljše kot druge. Sicer obstaja določena korelacija anomalij in nizke frekvence trgovanja, vendar ta in podobne karakteristike povezav nimajo velike informacijske vrednosti. Nasploh pa je težko najti dopustno (tako, ki ne precenjuje) hevristiko.

Obstaja možnost, da bi hevristiko uporabljali med delovanjem iskalnika in sicer na način, ki bi dajal večjo težo takim povezavam, ki so vsebovane v že najdenih pozitivnih ciklih (neke vrste lokalizacija). Z uporabo dinamičnega programiranja bi se mogoče ta način celo obnesel, torej bi najprej odkrival bolj verjetne pozitivne poti. Z najprej je mišljeno, da bi pri ocenjevanju začel s potmi, ki imajo večjo verjetnost, da so ugodne. Seveda pa nimamo razloga, da bi po prvi ocenjeni poti algoritem ustavili. Takega preiskovanja se nisem posluževal, ker tu lahko uporabimo izčrpno, ki nam da popoln rezultat v zadosti hitrem času.

2.5 Algoritem za iskanje elementarnih ciklov

To poglavje opisuje tako imenovan Johnsonov algoritem iz leta 1975[2]. Ta algoritem na nek način izboljšuje starejša algoritma Tiernana[10] in Tarjana[9] in je od njiju hitrejši, ker vsako povezavo obravnava največ dvakrat med obravnavanim in naslednjim ciklom.

2.5.1 Definicije

Definicije so v večini povzete po članku, v katerem je Donald B. Johnson prvič objavil v tej nalogi uporabljen algoritem. Kjer se definicije ali imena v znanstveni sferi razhajajo, bo to posebej omenjeno.

Vozlišče je osnovni element grafa in lahko vsebuje dodatne informacije ali pa je preprosta točka.

Povezava ali rob ali veja označuje odnos med vozlišči. Lahko nosi tudi dodatne informacije, kot je na primer smer. Taka povezava je usmerjena. Če smer ni definirana, govorimo o simetrični povezavi.

Usmerjen graf je graf G = (V, E), ki sestoji iz neprazne in omejene množice vozlišč V in urejene množice neponavljajočih parov E, ki predstavljajo povezave. V grafu je n vozlišč in e povezav.

Pot v grafu G je zaporedje vozlišč $p_{vu} = (v = v_1 = v_2, \ldots , v_k = u)$ tako, da $(vi, vi+1) \in E$ za $1 \leq i < k$.

Cikel je pot, v kateri sta prvo in zadnje vozlišče identični.

Pot je elementarna, če se nobeno vozlišče v njej ne pojavi dvakrat.

Cikel je elementaren, če se nobeno vozlišče razen prvega in zadnjega ne pojavi dvakrat. To implicira, da ne more biti podvojena niti povezava.

Dva elementarna cikla sta ločena ali disjunktna, če eden ni ciklična ali krožna permutacija drugega. To je eno glavnih dejstev, ki jih izkoriščamo v tej nalogi. Razlaga se nahaja v poglavju 3.1.5.4.

Induciran podgraf H je graf, ki ima natanko enake povezave med vozlišči, ki so podmnožica grafa G, kot jih imajo pripadajoča lastna vozlišča grafa G.

Močno povezana komponenta grafa je podgraf, kjer so vsa vozlišča dosegljiva iz vseh drugih vozlišč v tem podgrafu.

2.5.2 Delovanje Johnsonovega algoritma

Ker je algoritem v osnovnih delih enak kot ostali, bom opisal samo njegove bistvene značilnosti in izboljšave.

Johnsonov algoritem začne graditi cikle v začetnem vozlišču in nadaljuje po podgrafu, induciranem iz tega vozlišča ter vozlišč, urejenih po določenem ključu. Tako je izhod urejen po velikosti. Za preprečevanje podvojenih poti je vozlišče, ki je dodano v neko elementarno pot z začetkom v določenem vozlišču, blokirano. Blokirano ostane toliko časa, dokler vsaka elementarna pot iz trenutnega vozlišča do začetka ne seka trenutne poti v vozlišču, ki ni začetno. Druga bistvena izboljšava glede na prejšnje algoritme je, da vozlišče ne postane začetno vozlišče za gradnjo elementarnih poti, razen če ni zadnje vozlišče v vsaj enem od elementarnih ciklov. Tako se izogne velikemu delu neuspešnih iskanj.

Algoritem sprejme graf, predstavljen s strukturo sosednosti, ki jo sestavlja seznam povezav, te pa so predstavljene z izhodiščnim vozliščem in vozliščem na katerega povezava kaže. Algoritem predpostavlja, da so volišča predstavljena z celimi števili od 1 naprej.

Algoritem začne z iskanjem elementarnih poti v prvem vozlišču. Vozlišča trenutne elementarne poti se shranjujejo na skladu. Vozlišče se doda elementarni poti s klicem metode »CIRCUIT« in je izbrisano po vrnitvi iz tega klica. Ko se vozlišče doda, se ga označi kot blokiranega, da ne more biti uporabljen dvakrat na isti poti. Po vrnitvi iz klica, ki ga blokira, ni nujno, da se ponovno odblokira. Omogočenje je vedno dovolj zavrto, da sta dve omogočenji istega vozlišča ločeni z vrnitvijo novega cikla ali vrnitvijo v glavno proceduro. Celoten potek in dokaz o pravilnosti je na voljo v [2].

2.5.3 Časovna zahtevnost Johnsonovega algoritma

Johnsonov algoritem je bil ob izidu najhitrejši, razen za trivialno majhne grafe. Hiter je predvsem zaradi optimiziranega iskanja močnih komponent v grafu in dinamičnega

zmanjševanja števila obravnavanih vozlišč z uporabo zastavice za blokiranje teh vozlišč. To blokiranje ni časovno zahtevno, saj se blokada kliče le enkrat za posamezno začetno vozlišče, enako pa velja tudi za odblokiranje. Zaradi dejstva, da se prostor delovanja neposredno prekriva s prostorom Tarjanovega algoritma, le brez blokiranih poti, ki ne prinašajo plodnih rezultatov, se ta algoritem še posebej izkaže v redkejših grafih. To so grafi, ki imajo manjše število povezav.

Števila elementarnih ciklov ni mogoče preprosto opisati z enačbo, ker je odvisno od topologije grafa. Lahko pa si pomagamo z nekaterimi specifični grafi, za katere je ta problem rešen. Tako na primer za poln graf, to je graf, kjer je vsako vozlišče povezano z vsakim drugim, vemo število elementarnih ciklov, ki jih lahko opišemo z enačbo iz članka [2]:

$$\sum_{i=1}^{n-1} \binom{n}{n-i+1} (n-i)!$$

Iz tega lahko vidimo, da v najslabšem primeru, to je v primeru zelo povezanega grafa lahko število elementarnih ciklov z n narašča hitreje kot 2^n.

V Johnsonovem algoritmu je porabljen čas med opisom določenega cikla in naslednjega za njim največ **O(n + e)**, kjer je **n** število vozlišč, **e** pa število povezav. Zahtevnost ni večja niti pred prvim, niti za zadnjim obravnavanim vozliščem. Iz tega sledi, da za celoten graf porabi **O((n+e)(c+1))** časa, kjer je **c** število elementarnih ciklov. Poraba pomnilniškega prostora je **O(n + e).** Kot je že bilo omenjeno, je tak rezultat oziroma izboljšava glede na Tiernanov algoritem dosežena predvsem zaradi tega, ker med opisom kateregakoli cikla in opisom njegovega naslednika določeno vozlišče algoritem obravnava največ dvakrat.

Alternativni algoritmi za odkrivanje elementarnih ciklov in njihove časovne zahtevnosti so opisani v poglavju 4.1.

Poglavje 3 Opis programa

3.1 Zgradba programa

Program je izdelan kot Python modul v 64-bitnem okolju jezika Python 3.4. Je objektno zasnovan in uporablja ideje razširljivosti, modularnosti, skalabilnosti in preglednosti.

Projekt je zasnovan tako, da ga je mogoče uporabljati čim bolj neodvisno. Kjer je bilo mogoče, sem uporabil knjižnice, ki so vključene v referenčno namestitev jezika. Zunanji paketi, ki sem jih uporabil, so navedeni v podpoglavju 3.1.3 in so prosto dostopni v Pip repozitorijih ali kot že zgrajeni namestitveni programi za operacijski sistem Windows, na katerem sem delal.

Razširljivost je zagotovljena predvsem z zasnovo razredov. Vsi so izvedeni z mislijo, da je mogoče dodajati nove funkcionalnosti. V razred ApiParser je na primer preprosto mogoče dodajati povezave za nove borze. Razred Commodity lahko konstruiramo z najrazličnejšimi možnimi tipi vrednostnih papirjev, dragih kovin, valut ali surovin. Razred Exchange sprejme različne vrste borz in menjalnic, razred Relations pa različne povezave med njimi ter različne načine provizij in razmika med trgovalnimi cenami.

Modularnost je mišljena s tem, da je v program mogoče preprosto vključiti na primer modul za napovedovanje vrednosti ali pa izris vseh povezav med trgovalnimi elementi.

Ozki grli programa sta pridobivanje ažurnih podatkov iz borz (linearen problem, natančneje opisan v poglavju 3.1.4) in ugotavljanje elementarnih ciklov. Vendar program zaradi razmeroma dobrega algoritma za iskanje ugodnih trgovanj lahko sprejme nekaj sto različnih trgovalnih elementov, ki lahko skupaj tvorijo do nekaj tisoč elementarnih ciklov. Dejansko to pomeni, da lahko z njim predstavimo vse trgovalne elemente, ki jih poznamo in se z njimi aktivno trguje, ob pogoju, da niso preveč močno povezani (da ni mogoče z vsakim trgovati z veliko drugimi elementi).

Posebno omembo si zasluži obravnava vseh števil pri trgovanju v pravilni decimalni obliki, kot je to v navadi v finančnih aplikacijah, in ne z uporabo privzetega tipa za zapis s plavajočo vejico *float*. Razlog za to je, da nočemo izgubiti natančnosti, saj se pri takem trgovanju napake lahko hitro seštejejo v prevelike. Podrobneje je to opisano v poglavju 3.1.2.

3.2 Razredi programa

Posamezni razredi predstavljajo osnovne gradnike v obravnavani domeni in implementirajo metode za njihovo manipulacijo.

3.2.1 ApiParser

Je prvi razred ali začetek cevovoda, kjer program sprejme podatke. Razred vsebuje zelo pomembno metodo *populate_relations(self, relation, ONLY_SEARCH_EXCHANGES, INFO)*, ki je vmesni sloj ali ovoj okoli različnih vmesnikov borz.

Razred je dejansko vmesnik med aplikacijskimi vmesniki borz in programom. Vse zunanje aplikacijske vmesnike namreč poenoti oziroma ovije v ustrezno obliko, da je iz njih mogoče črpati podatke v enotnem formatu. S temi podatki napolni strukture relacij (graf) med trgovalnimi elementi in seveda ustvari elemente razreda Commodity. V ta namen se neposredno povezuje na vmesnike borz, ki so v večini izvedeni kot storitve REST, ali pa za to uporablja dodatne, za to pripravljene vmesnike.

Neposredno iz borz je mogoče dobiti javne podatke z omejeno časovno ločljivostjo. V primeru, da želimo na borzi trgovati, torej postavljati ponudbe, kupovati, prodajati, preverjati svoj portfolio, je potrebna varna povezava na storitev te borze. Ta je izvedena z uporabo zasebnega ključa, ki ga uporabnik prejme na borzi. Ključ je shranjen v tekstovni obliki, v poljubni datoteki, bere se le ob vzpostavljanju nove povezave. Da je to potrebno storiti čim manjkrat, ustvarjeno povezavo recikliramo (t.i. »connection pooling«) za več poizvedb in tako pohitrimo delovanje.

Sicer pa ustvarjanje povezav ni ozko grlo tega razreda. Večji problem so vsiljene časovne omejitve borz in zakasnitve na prenosnem omrežju. Ti dve težavi lahko v določenih primerih zelo omilimo z uporabo tehnologije »WebSocket«, ki jo podpirajo nekatere borze. Več o tem v poglavju 4.2.

Razred si pomaga z modulom *Decimal* za upravljanje pravih decimalnih števil in z različnimi moduli zunanjih aplikacijskih vmesnikov.

3.2.2 Commodity

Razred *Commodity* opisuje en trgovalni element in njegove lastnosti. Ta element je lahko vsaka valuta, vrednota ali surovina, s katero je mogoče trgovati. Najpogosteje so to denarne valute, kripto valute, delnice, obveznice, surovine, blago, terminske pogodbe, hipoteke ipd. Razred skrbi tudi za pravilen izpis enote in obravnavo v programu ter za način obravnave enote pri urejanjih.

Ne vsebuje povezav do drugih valut ali cen, ki jih moramo plačati za nakup druge valute. Prav tako ni opredeljen z borzo, je pa element določene borze. Ne vsebuje niti provizij za trgovanje. Relacije med trgovalnimi elementi hrani za ta namen predviden razred *Relations*.

3.2.3 Exchange

Razred Exchange opisuje eno trgovalno borzo in njene lastnosti. Borze se lahko ukvarjajo s poljubnim načinom trgovanja, obračunavanjem provizij in izvedbo trgovanja. Vsebuje tudi trgovalne pare, ki se na njej trgujejo in skrbi za izpis ter urejanje, poleg tega pa vsebuje načine prenosa sredstev za trgovanje.

3.2.4 Main

To je osnovni razred, kjer se program zažene in začne izvajati. Upravlja z celotnim cevovodom, skozi katerega potujejo podatki. Vsebuje osnovne nastavitve in parametre za delovanje, kot so seznam borz, po katerih je potrebno iskati, razne omejitve, tip iskanja, če je na primer potrebno trgovalne dogodke izvajati verižno ter ali naj program izpisuje informacije o delovanju. V njem je mogoče tudi omejiti trgovalne elemente, ki jih želimo uporabiti.

Razred najprej ustvari instanco *Relations*, ki hrani vse relacije v obliki usmerjenega grafa. Potem z ustreznimi argumenti kliče metode razreda *ApiParser*, ki pridobi, predela in vstavi podatke relacij in trgovalnih entitet v graf. Na tej točki je pridobljene podatke o relacijah mogoče tudi shraniti v serializirano obliko v datoteko **.pickle*, da nam ni vedno potrebno prenašati novih podatkov, če to ni nujno, s čimer se zelo pohitri prvi del programa. Ko pridobimo vse željene podatke, jih začnejo obdelovati metode razreda *PathFinder*, ki v njih z uporabo elementarnih ciklov najdejo ugodne trgovalne povezave. Če je potrebno, ta razred zgrajeni graf relacij tudi izriše, na koncu pa izpiše ugodne relacije glede na podane pogoje.

3.2.5 PathFinder

Ta razred je jedro aplikacije, saj skrbi za iskanje in ocenjevanje pozitivnih trgovalnih ciklov. To stori tako, da iz podanega grafa relacij in trgovalnih elementov izlušči množico elementarnih ciklov z uporabo Johnsonovega algoritma. Te cikle pretvori v zaporedne pare trgovalnih elementov, kar predstavlja trgovalne dogodke. Trgovalne dogodke združi v verigo in celotno verigo ob upoštevanju tržne cene, provizij in globine trga oceni. Tako ocenjene verige oziroma cikle vrne, če ustrezajo podanim kriterijem.

Razred skrbi tudi za informativni izris podatkovne strukture, v kateri hranimo vse podatke. To je instanca *Relations*, ki vsebuje instanco *DiGraph* knjižnice *Networkx*. Za izris uporablja *pyplot* iz knjižnice *matplotlib*.

3.2.6 Relations

Glavna naloga razreda *Relations* je hramba podatkov. Za to sem najprej uporabljal podatkovno strukturo po meri, kasneje pa instanco *DiGraph* knjižnice *Networkx*. Gre za usmerjen graf, v katerem vozlišča predstavljajo instance razreda Commodity, povezave pa pomenijo, da je v neko smer mogoče trgovati. Povezave nosijo še dodatne informacije, kot so borze, provizije, časovni žigi ter knjige ponudb in povpraševanj, z globinami le teh za posamezne borze.

Metode razreda so namenjene operacijam nad grafom, kot je dodajanje povezav, relacij, trgovalnih elementov … Preprečujejo, da bi nehote pokvarili podatkovno strukturo, ki je sestavljena iz glavnega slovarja slovarjev in drugih pomožnih struktur, kot so množice ključev in podobno.

Za upravljanje s časovnimi žigi uporabljam modul *datetime*.

3.2.7 Test

Modul *test* je namenjen testiranju. Ker ta del ne sodi v ožje jedro te naloge, metod nisem striktno enotsko testiral, sem pa delal sprotno testiranje najbolj problematičnih delov kode.

3.3 Decimal, float, zaokroževanje števil

V aplikacijah, kjer je potreba po natančni predstavitvi decimalnih števil, kamor nedvomno spadajo tudi finančne aplikacije, je potrebno posebno pozornost posvetiti podatkovnemu tipu, ki ga bomo uporabljali. Znano je, da zapisi s plavajočo vejico v binarnem formatu lahko točno predstavijo decimalna števila le na določeno število mest natančno, kar je odvisno od dolžine takega zapisa. Ta fenomen izhaja iz dejstva, da so ta števila fizično predstavljena v binarnem formatu, ki je tako široko uporabljan zato, ker lahko z isto dolžino predstavimo tako mikroskopske kot tudi astronomske razsežnosti. Za to žrtvujemo ločljivost, s katero lahko predstavimo racionalna števila. Vseh racionalnih števil namreč s tem formatom ni mogoče zapisati. Bolj kot se namreč oddaljujemo od plavajoče vejice, redkejša so števila, predstavljena v formatu *float.* Z njimi ni vedno mogoče opisati natančne vrednosti, ker poljubno število lahko predstavimo le z najbližjim takim številom, ki obstaja.

Zaradi tega bi torej račun 0.1 + 0.2 == 0.3 bil napačen. To rešujemo tako, da namesto natančnosti žrtvujemo hitrost izvajanja in velikost zapisa. V aplikaciji je v ta namen uporabljen tip *Decimal* iz modula *decimal*.

Pri tem pa je še vedno potrebno paziti na več stvari, kot so mesta pri zaokroževanju in kreiranje instance *Decimal*. Če jo ustvarimo s konstruktorjem tega razreda iz tipa *float* (v resnici je *float* v jeziku Python tip *double* iz jezika C) na primer tako: *provizija = Decimal(0.002)*, smo že vpeljali napako. Tu provizija ni točno 0.002, ampak: 0.002000000000000000041633634234433702658861875534057617187 5. Tega se rešimo tako, da uporabljamo posebno metodo, ki pametno zaokrožuje, ali pa težavo rešimo še bolje in decimalno število kreiramo iz niza znakov, na primer tako: *provizija = Decimal("0.002").*

Podatki iz borz so vedno že zapisani v decimalnem formatu, ki ga tudi sicer interno uporabljajo.

Zavedati se je potrebno, da s tem, ko v celotnem programu za računanje uporabljamo pravilna decimalna števila, žrtvujemo hitrost tipa *float* ali *double*, katerih preračunavanje je izvedeno strojno. Ustvarjanje nove instance tako traja 2x dlje kot ustvarjanje nativnih tipov. Računanje z njimi pa lahko traja tudi 20x dlje v primeru deljenja.

3.4 Kniznjice

V aplikaciji sem uporabil sledeče knjižnice:

- numpy – osnovna knjižnica za znanstveno računanje in delo s števili, vektorji, matricami;
- python dateutil – knjižnica za delo s časom, funkcije za manipuliranje časa;
- matplotlib – knjižnica za risanje, narejena po vzoru funkcije *plot()* iz programa Matlab;
- graphwiz – knjižnica za izris grafov in njihovo postavitev v prostoru;
- pyparsing – knjižnica za delo z besedili;
- python-igraph – knjižnica, narejena v jezikih C in C++, namenjena analizi grafov;
- pycairo – grafična knjižnica s povezavami v Pythonu;
- graph-tool – knjižnica za analizo grafov, ki temelji na knjižnici Boost Graph Library;

- networkx – knjižnica za analizo grafov, v celoti narejena v jeziku Python;
- datetime – knjižnica za delo s časom;
- decimal – knjižnica, ki vpelje nov precizen tip Decimal;
- pickle – modul za serializacijo objektov v jeziku Python;
- hashlib – knjižnica za delo z varnimi razprševalnimi funkcijami;
- requests – HTTP knjižnica, ki dopolni urllib;
- hmac – implementacija razprševanja ključev za avtentikacijo po standardu RFC 2104;
- time – modul za delo s časom;
- json – knjižnica za delo z JSON objekti;
- re – knjižnica za delo z regularnimi izrazi;
- http.client – modul, ki definira razrede, za klientovo stran http in https protokolov;
- warnings – modul za delo z opozorili; za uporabo v primerih, kjer izjema ni potrebna;
- html.parser – osnovni modul za razčlenjevanje http formata;
- urllib – visokonivojski vmesnik za pobiranje podatkov preko svetovnega spleta.

3.5 Aplikacijski vmesniki za borze

Aplikacijski vmesniki borz se lahko zelo razlikujejo. Večinoma so izvedeni kot REST storitve, včasih pa celo s tehnologijo »WebSocket«, ki je standardiziran dvosmerni protokol, ki se naslanja na eno povezavo TCP. Preko slednje borze običajno dovoljujejo več prometa in omogočajo hitrejše dostopne čase. Tega za REST storitve ne moremo reči, saj so v večini primerov strogo omejene na število poizvedb v določenem času in na čas med poizvedbami. Pri razvoju aplikacije sem zelo hitro naletel na te omejitve in si zaradi prevelikega števila poizvedb celo zaslužil blokado svojega naslova IP.

Vsak aplikacijski vmesnik je drugačen. Zaradi tega sem ustvaril razred, ki vse pretvori v za mojo aplikacijo smiselno obliko. V tem razredu je izvedeno vse razčlenjevanje in pretvarjanje originalnih podatkov. Poskrbeti je bilo potrebno za vrsto pretvorb in poenotenj, saj na primer

za isto valuto obstaja več poimenovanj ali so prodajne in nakupne cene podane le iz vidika ene valute. Pri provizijah pa je potrebno upoštevati vrsto parametrov; vsak trgovalni par ima lahko različno provizijo, ki se lahko obračunava tudi glede na pretekli volumen trgovanja ali pa v točno določeni valuti.

Tu se vzpostavi tudi varna komunikacija, ki je potrebna za poslovanje z osebnim računom na borzi. Avtenticiramo se z zasebnim vnaprej izmenjanim ključem. Po avtentikaciji lahko dostopamo tudi do podatkov aplikacijskega vmesnika, ki so namenjeni izključno nam.

3.6 Uporabljeni algoritmi, razmislek

3.6.1 Idealne in slabše poti

Idealne poti, ki si jih želimo, niso daljše, kot je nujno potrebno za dosego željenega učinka. To je največkrat pot dolžine tri, lahko pa se zgodi, da je pot daljša. Taka pot je veljavna oziroma dobra le, če ne obstaja nobena pot, ki bi lahko isti učinek dosegla četudi z večjim volumnom trgovanja. Če se spremeni cena kateremu koli trgovalnemu elementu, trgovalna pot namreč nima več izračunane vrednosti, dovolj pa je že, da se zniža globina ponudbe kateregakoli člena na manjšo vrednost, kot je vrednost globine prejšnjega najnižjega člena. Z daljšanjem trgovalne verige se daljša čas izvajanja transakcij in s tem možnost za spremembe na trgu. Lahko se celo zgodi, da mi sami s prvimi transakcijami preveč vplivamo na trg in s tem spremenimo cene naslednjim. Poti dolžine dva lahko izločimo, ker na zdravih borzah ne morejo prinašati dobička. Če si predstavljamo, da iz valute A zamenjamo v valuto B in ob tem upoštevamo še provizijo, se namreč ne more zgoditi, da bi imeli na koncu od tega zaslužek.

3.6.2 Vsi cikli v grafu

Število vseh ciklov v grafu se da določiti zelo hitro z znano formulo[2]: $2^{(e - v + 1)} - 1$, kjer so vozlišča grafa označena z 'v' in robovi s črko 'e'. Lahko si predstavljamo, da število ciklov eksponentno raste z razmerjem števila povezav in števila vozlišč. Več kot je torej povezav na eno vozlišče, več je ciklov[3]. V praksi to pomeni, da sem samo na eni borzi, ki trguje s 25 različnimi trgovalnimi pari, eksperimentalno določil oziroma oštevilčil[1] več sto milijonov različnih ciklov. Odkrivanje uspešnih poti na toliko ciklih je nepraktično in tudi neuporabno, saj so cikli daljši, kot je nujno potrebno. Tako dolgi cikli pa so izjemno nezaželeni, ker je tako verigo transakcij v praksi težko izvesti prej, kot se pogoji na trgu spremenijo.

3.6.3 Zaprti sprehod po grafu

Zaprti sprehod po grafu je pot na grafu, ki se začne in konča v istem vozlišču. Terminologija se tu razlikuje, vendar na ta primer to ne vpliva. Taki sprehodi nam ne koristijo, saj lahko večkrat obiščejo isto vozlišče, kar dejansko pomeni, da s tem ustvarijo dva cikla. Vozlišče, ki ga obiščemo dvakrat, si lahko predstavljamo kot središče osmice. Isti ali večji učinek (zaradi nižje omejitve globine trga; en del ima zagotovo najmanj enako globino kot drugi, skupna globina pa je vedno manjša ali enaka od obeh) dosežemo z razdelitvijo takega sprehoda na dva dela. Ob tem pa smo cikel še skrajšali.

3.6.4 Elementarni ali preprosti cikli

Cikel je elementaren, če se nobeno vozlišče razen prvega in zadnjega ne pojavi dvakrat. To implicira, da nobena povezava ne more biti podvojena. Po drugi definiciji pa so to cikli, ki nimajo niti zank (vozlišč, ki kažejo nase) niti ponovljenih robov.

Dva elementarna cikla sta ločena ali disjunktna, če eden ni ciklična ali krožna permutacija drugega.

Iz teh dveh definicij lahko vidimo, da so to točno cikli, ki so najkrajši možni, obenem pa pokrivajo vse mogoče poti. Krožna permutacija je za nas v bistvu isti cikel, le da pot začnemo z drugo valuto. Končna vrednost takih dveh permutacij je torej enaka. Odveč so le cikli dolžine dva, ki jih je toliko, kolikor obojestranskih povezav med vozlišči obstaja v grafu. Teh je glede na število ciklov in kompleksnost iskanja le-teh zanemarljivo majhen delež in jih preprosto izločimo. Lahko pa jih pustimo in nam služijo za preverjanje napak pri postavljanju ponudb na trg.

Kompleksnost iskanja takih ciklov je opisana v poglavju 2.5.3.

3.6.5 Filtriranje nepotrebnih

Preden sem za iskanje ugodnih poti uporabljal elementarne cikle, sem poskušal najti način, da bi pridobil dobre cikle (glej 3.5.1) iz množice vseh. V ta namen sem si zamislil algoritem, ki najprej razvršča cikle po dolžini (najkrajši so namreč za naše potrebe vedno idealni). Drugi korak je iteriranje po razvrščenih ciklih, z začetkom pri ciklih dolžine 3. Vozlišča teh ciklov dodajamo v množico že videnih vozlišč v primeru, da je v tem ciklu vozlišče, ki ga še nismo videli. Na ta način filtriramo nepotrebne daljše cikle. Pri tem obstaja izjema. V primeru, da cikel obdaja nabor ciklov, ki smo jih že sprejeli, bo padel skozi sito, kljub temu da obstaja

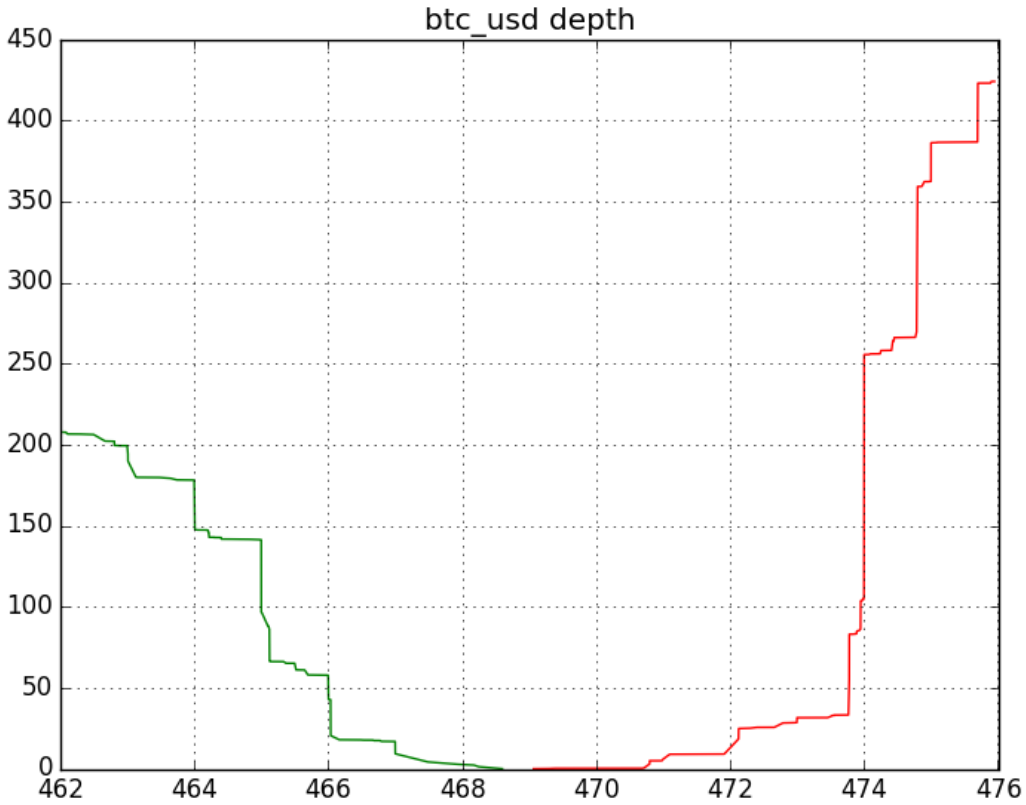
možnost, da gre za dober cikel. Če hočemo ta problem rešiti v polnosti, je potrebno vsak zavržen cikel testirati še za ta primer. To močno dvigne časovno zahtevnost tega algoritma. Na srečo sem kmalu odkril veliko bolj učinkovit Johnsonov algoritem in tega opustil.

3.6.6 Ciklični zapis permutacij

Če bi mi čas dopuščal, bi se lotil razmisleka, kako tehniko cikličnega zapisa permutacij uporabiti na vozliščih grafa, saj se mi zdi, da bi lahko dobljene transpozicije uporabil kot nekakšen filter pri iskanju ciklov, ki jih potrebujem.

3.6.7 Globina knjige naročil

Globina knjige naročil so dejansko vsi podatki o naročilih za trgovanje na borzi. Delijo se na prodajne in nakupne, sestavljajo pa jih pari cen in količin enot v naročilu. Če izrišemo vse, dobimo spodnjo sliko.



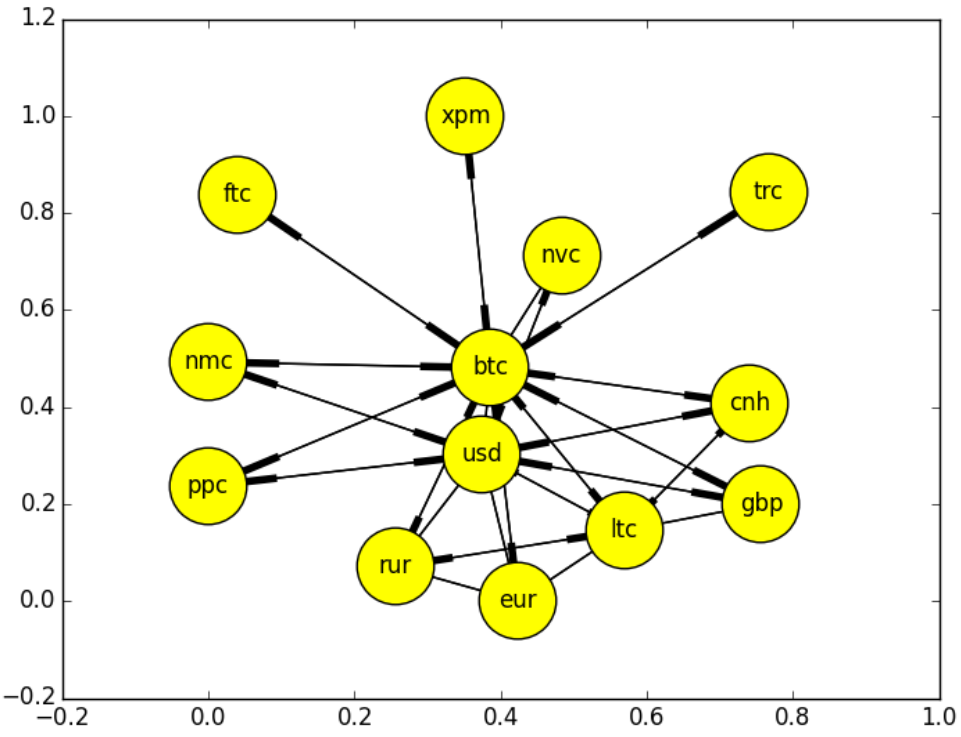
Slika 3.1: Knjiga naročil na dan 12. 9. 2014 iz podatkov borze Btc-e

Na abscisi je cena za eno enoto, na ordinati pa količina enot, ki so na voljo. Površina pod grafom predstavlja ceno, ki jo je potrebno plačati za nakup vsega volumna na določenem intervalu.

Tržna vrednost je vrednost, kjer se nakupna in prodajna naročila skoraj srečajo, razmak (»spread«) pa je razlika med trenutno nakupno in prodajno ceno. To je vrednost, ki jo moramo v enem ciklu premagati (za vsako transakcijo v ciklu), da je cikel pozitiven.

3.6.8 Izris grafa

Za izris grafa uporabljam knjižnico matplotlib v povezavi s knjižnico networkx. Na sliki je izrisan primer relacij trgovalnih elementov na dveh borzah. Gre za usmerjen graf, smer povezave je pri cilju označena z odebeljeno črto. Na tem grafu vidimo, da je skoraj z vsemi valutami mogoče trgovati v obe smeri. Za postavitev grafa je uporabljen tako imenovan »shell_layout«.



Slika 3.2: Usmerjen graf(odebelitve kažejo smer) iz podatkov dveh borz.

3.6.9 Serializacija - pickle

Zaradi razmeroma počasnih aplikacijskih vmesnikov na borzah in količine poizvedb, ki jih je potrebno narediti za gradnjo takega grafa, uporabljam predpomnenje v obliki datoteke pickle. Sploh pri razvoju ažurni podatki niso tako pomembni kot hitrost izvajanja. Shranjevanje se zgodi po gradnji grafa. Na ta način se izognem prevelikemu številu poizvedb, ki imajo za posledico onemogočenje naslova IP in pohitrim vsako naslednje izvajanj programa iz približno 30 sekund na 2 sekundi.

3.6.10 Največje težave pri implementaciji

Največ težav pri izdelavi programa mi je predstavljalo neprestano miselno sledenje nakupnim in prodajnim tečajem, ki so na voljo v množici različnih formatov in zornih kotov. Eno enoto lahko menjaš za n enot ali obratno, ob tem pa imaš dva seznama cen naročil, nakupnega in prodajnega. Če se tu zmotiš, napaka zaradi majhnih razlik med njimi ne bo vedno vidna takoj, se bo pa ob trgovanju z napačnimi predpostavkami zelo poznala na debelini denarnice.

Druga večja težava je bila obravnava provizij. Kot sem že omenil, namreč nekatere borze količino provizij določajo dinamično – glede na volumen tvojega preteklega trgovanja.

Poglavje 4 Pohitritve, izboljšave, mogoče nadaljevanje

Ideje za pohitritev in nadaljnji razvoj so razporejene po pomembnosti oziroma stopnji pohitritve, ki jo lahko prinesejo.

4.1 Boljši in nepopolni algoritmi za iskanje elementarnih ciklov

V literaturi lahko najdemo več algoritmov za odkrivanje elementarnih ciklov v usmerjenih grafih. V prvih takih algoritmih, kot sta Tiernanov[10] in podoben Weinblattov, je potreben čas za iskanje vsakega novega cikla lahko eksponenten glede na velikost samega grafa[2]. To izredno slabo lastnost je izboljšal Tarjanov algoritem[9], ki za to porabi največ O(n*e) časa, kar pomeni v najslabšem primeru O(n*e(c+1)) za celotni graf. Ehrenfeucht, Fosdick in Osterweil so podali podoben algoritem, ki deluje v enakem časovnem okviru. Ti algoritmi ne izkoriščajo dejstva, da si lahko pomagamo z razbijanjem grafa na njegove močne komponente in ga tako vsaj delno razbijemo na manjše podprobleme. Johnsonov algoritem za izdelavo vsakega naslednjega elementarnega cikla potrebuje samo O(n+e) časa, za cel graf pa O((n+e)(c+1)). Omenjen opis je povzet po [2].

Za iskanje elementarnih ciklov poznamo še druge algoritme z različnimi kompleksnostmi. Če upoštevamo časovno zahtevnost, v najslabšem primeru lahko najbolj znane razporedimo po takem vrstnem redu:

1. Tiernan [10] - $O(V.const^V)$
2. Tarjan [9] - O(VEC)
3. Johnson [2] - O(((V+E)C)
4. Szwarcfiter in Lauer [7] - O(V+EC)

Manj znan Weinblattov algoritem je glede tega podoben kot Tiernanov. Ehrenfeuchton, Fosdickov in Osterweilov algoritem pa realizirajo isto zahtevnost kot Tarjanov.

Vendar so take zahtevnosti dosežene le pri grafih s posebno neobičajno strukturo. V praktičnih primerih algoritmi z večjo kompleksnostjo večkrat prehitijo take z nižjo v najslabšem primeru.

Razlog za to so po navadi večji »administrativni stroški« algoritmov z nizkimi največjimi kompleksnostmi. Po navadi pa imajo tudi večjo porabo pomnilnika. Ti algoritmi imajo sicer enako O(V+E). V predstavlja velikost vozlišč in E velikost povezav.

Če se želimo zadovoljiti z nepopolnim iskanjem, lahko uporabimo hitrejše algoritme[5], ki iščejo samo elementarne cikle točno določene dolžine, toda realizirajo bistveno boljše časovne zahtevnosti, če je obravnavan graf relativno redek ali relativno degeneriran. Za elementarni cikel dolžine 3 bi tako porabili $E^{1.41}Ed(G)$, za cikel dolžine 4 pa $E^{1.5}Ed(G)$ v usmerjenih grafih[5]. Pri neusmerjenih je ta zahtevnost še nižja[5].

4.2 WebSocket

REST poizvedbe so relativno počasne, tako zaradi tehničnih razlogov kot zaradi vsiljenih omejitev na borzah. Pri tem si sicer pomagamo s tem, da preko ene povezave pridobivamo več podatkov in tako prihranimo ponovno vzpostavljanje, resnične izboljšave pa prinaša uporaba tako imenovane »WebSocket« povezave, kjer lahko borza sama pošilja sveže podatke brez zahtevkov klienta. Take povezave na žalost ne omogočajo vse borze.

Pohitritev v tem primeru je linearna. Dodatna prednost so bolj ažurni podatki in njihova večja časovna ločljivost.

4.3 Knjižnica za graf, izvedena v jeziku C

Trenutno uporabljana knjižnica networkx, v kateri je izveden graf, je implementirana v programskem jeziku Python. Obstajajo odlične knjižnice, izvedene v programskem jeziku C in C++, na primer graph-tool ali igraph.

Pohitritev bi bila malo večja kot linearna. Dodatna prednost je, da omenjeni knjižnici dajeta na voljo več funkcionalnosti. Slabost pri knjižnici igraph je neintuitiven api. Graph-tool pa se lahko dodatno pohvali z večnitnostjo.

4.4 Večnitnost

Z uporabo paralelizacije bi lahko linearno pohitrili nekaj delov programa, predvsem pridobivanje in vstavljanje podatkov, ki ga ju mogoče skoraj perfektno razdeliti v vzporedne niti. Prav tako bi se dalo odlično pohitriti ocenjevanje ciklov, saj to brez težav počnemo vzporedno, ker med sabo niso povezani.

4.5 Pasivno/aktivno trgovanje

Večje dobičke bi si lahko obetali, če bi namesto pasivnega trgovanja poskušali aktivno dodajati svoje ponudbe na trg. Tu je potrebno nekaj razmisleka o našem vplivu na trg. Izkoriščamo lahko dejstvo, da nakupna in prodajna cena nista enaki. V ozkem oknu med njima lahko ob pogoju, da je trg dovolj likviden in frekvenca trgovanja dovolj visoka, s svojo ponudbo dobimo boljšo ceno kot s sprejemom tuje ponudbe.

4.6 Agregatorji borznih indeksov

Obstajajo zbirniki podatkov iz borz, kjer bi lahko dobili zelo velik nabor novih podatkov, vendar hkrati tudi nekaj novih težav. Prva je, da moramo v tem primeru nekomu zaupati, da so podatki res pravilni. Druga zelo očitna pa je počasnost in nizka frekvenca osveževanja podatkov, zaradi česar izgubimo ažurnost.

4.7 Trgovanje s posedovanjem več trgovalnih elementov

Ideja tega načina je, da trgovanja ne začnemo z enim trgovalnim elementom in z namenom, da bi imeli na koncu trgovanja več tega elementa, ampak začnemo z neko vrednostjo vseh trgovalnih elementov. Ko najdemo ugoden cikel, lahko v tem primeru naenkrat prodamo vse udeležene trgovalne elemente in si tako zagotovimo pravo ponujeno ceno; ne tvegamo, da bi naš vpliv prvih nakupov že vplival na naslednje nakupe na borzi. Tako razporejene valute se lahko ohranjajo na podobnih nivojih (če so vse povezave v ciklu podobno ugodne), kar pa po navadi ne drži, saj običajno določene valute »tečejo« v druge. Zaradi tega na koncu tvegamo izgubo, ker moramo vse elemente pretvoriti nazaj v željeno valuto. V primeru, da imamo veliko valute, za katero je ta pretvorba neugodna, lahko naredimo minus.

4.8 Uporabniški vmesnik in aplikacijski vmesnik

Aplikaciji bi lahko izdelali prijeten uporabniški vmesnik, ki bi tudi neveščim uporabnikom omogočal trgovanje na borzi.

Poleg tega bi lahko preko aplikacijskega vmesnika dali učinkovito zbrane podatke na voljo še drugim razvijalcem aplikacij.

4.9 Programerska optimizacija

Po navadi lahko v programiranju sprejemamo kompromise. Če algoritem porabi preveč pomnilnika, ga lahko predelamo tako, da bo časovno bolj potraten v zameno za nižjo porabo pomnilnika ali obratno. Tudi tu je tako. Pohitritev na področju časovne zahtevnosti in porabe pomnilnika lahko dosežemo, če žrtvujemo nekaj preglednosti in možnosti za analizo, s tem da ocenjevanje ciklov vgradimo neposredno v Johnsonov algoritem. Ta namreč vsak cikel prehodi vsaj enkrat. Zaradi tega ni potrebe po dodatnem shranjevanju in obdelavi teh ciklov.

Pohitriti je mogoče tudi ocenjevanje ciklov in sicer primerjanje sosednjih vozlišč. Sosednja vozlišča se namreč v različnih grafih pojavljajo večkrat. Z ustrezno optimizacijo bi lahko del tega računanja izvedli le enkrat; na ta način bi pri celotni kompleksnosti programa lahko dobili linearno izboljšavo.

Poglavje 5 Sklepne ugotovitve

Eksperimentalno lahko ugotovim, da v nalogi opisana arbitraža deluje in da je izvajanje trgovanj po krajših ciklih neprimerno lažje kot izvajanje daljših.

Znotraj ene borze izjemno redko najdemo pozitiven cikel. Glede na relativno visoko naključnost (1 - najboljši napovedni modeli) tržne cene med posameznimi trgovalnimi pari lahko z uporabo statistike z zelo velikim zaupanjem ugotovimo, da tako dejansko stanje ne sledi naravni porazdelitvi. To nam pove, da se arbitraža dejansko (vsaj na borzah, ki sem jih opazoval) aktivno izvaja. Druga zanimivost, ki jo lahko opazimo, pa je nivo izvajanja. Ko sem iz izračuna odvzel upoštevanje provizij, sem ugotovil, da je ugodnih ciklov kar nekaj – skoraj toliko, kolikor bi jih naravno moralo obstajati. To nam pove, da tisti, ki izvaja arbitražo (na primer lastniki borze), zelo verjetno lahko trguje brez ali z zelo nizkimi provizijami.

Ta pojav pri vrednotenju ciklov med različnimi borzami ni tako izrazit. Tu večkrat najdemo tudi velike razlike, ki jih lahko s pridom izkoriščamo. Edina težava, ki nastopi, je časovno dolg prenos sredstev iz ene borze na drugo. Ta težava spremlja predvsem fiat valute, kjer mednarodna nakazila trajajo relativno dolgo. Kripto valute se tu dobro izkažejo, saj so prenosi lahko instantni.

Literatura

[1]L.G.Bezem and J.van Leeuwen, Enumeration in graphs., Technical report RUU-CS-87-7, University of Utrecht, The Netherlands, 1987

[2] Donald B. Johnson. SIAM J. COMPUT. Vol. 4, No. 1, March , str. 77-84, 1975 Dosegljivo: http://www.cs.tufts.edu/comp/150GA/homeworks/hw1/Johnson%2075.PDF

[3] J. Katz, Notes on Complexity Theory, C.S. UMD, Lecture 23, Nov. 2011 Dosegljivo: http://www.cs.umd.edu/~jkatz/complexity/f11/lecture23.pdf

[4] P.Mateti and N.Deo, On algorithms for enumerating all circuits of a graph., SIAM J. Comput., maj 1978, str. 90-99.

[5] N. Alon, R. Yuster, U. Zwick, Finding and counting given length cycles, Algorithmica, March 1997, Volume 17, Issue 3, str. 209-223, Dosegljivo: http://www.tau.ac.il/~nogaa/PDFS/ayz4.pdf

[6] K. Paton, An algorithm for finding a fundamental set of cycles for an undirected linear graph, Comm. ACM 12, 1969, str. 514-51

[7] J.L.Szwarcfiter and P.E.Lauer, Finding the elementary cycles of a directed graph in O(n + m) per cycle, Technical Report Series, #60, Univ. of Newcastle upon Tyne, Newcastle upon Tyne, England, May 1974

[8] R.Tarjan, Depth-first search and linear graph algorithms., SIAM J. Comput. 1, 1972, str. 146-160

[9] R. Tarjan, Enumeration of the elementary circuits of a directed graph, SIAM J. Comput., feb. 1973, str. 211-216.

[10] J.C.Tiernan An Efficient Search Algorithm Find the Elementary Circuits of a Graph., Communications of the ACM, V13, dec. 1970, str. 722 - 726.